**UFR-IMA /Polytech' Grenoble**
**Ecole Universitaire d'Informatique de Grenoble**

**Travaux Pratiques d'Architecture Matérielle**

# Microprogrammation d'une machine de type processeur

## Présentation générale.

Les séances de TP sur le thème de la microprogrammation sont une suite directe du travail sur les circuits complexes (partie opérative, partie contrôle) et de l'étude des instructions des processeurs. Nous allons étudier la structure interne réelle d'une machine à exécuter des instructions, c'est-à-dire un processeur, et faire en sorte qu'elle soit capable d'interpréter et d'exécuter un jeu d'instructions donné.

Une telle machine peut être réalisée sous la forme Partie Contrôle/ Partie Opérative. C'est le cas par exemple pour le processeur 68000. La partie contrôle correspond à la réalisation d'un automate dont les sorties correspondent aux commandes de la PO (opérations à faire à un moment donné) et l'enchaînement des états traduit l'enchaînement dans le temps de ces opérations élémentaires.

Cet automate peut être réalisé de diverses manières:
- D'une façon dite câblée (Bascules + portes logiques)
- De manière dite microprogrammée comme nous allons nous y intéresser ici. La différence tient à la méthode de description de l'automate de contrôle et surtout à la réalisation matérielle de cet automate.

L'automate de contrôle qui représente l'algorithme d'interprétation des instructions sera écrit dans un langage que l'on appellera micro-langage. Un programme écrit dans ce micro-langage sera appelé microprogramme, il contiendra des lignes que l'on appellera micro-instructions (d'où le nom de microprogrammation). Chaque micro-instruction correspond en fait à un état d'un automate classique.

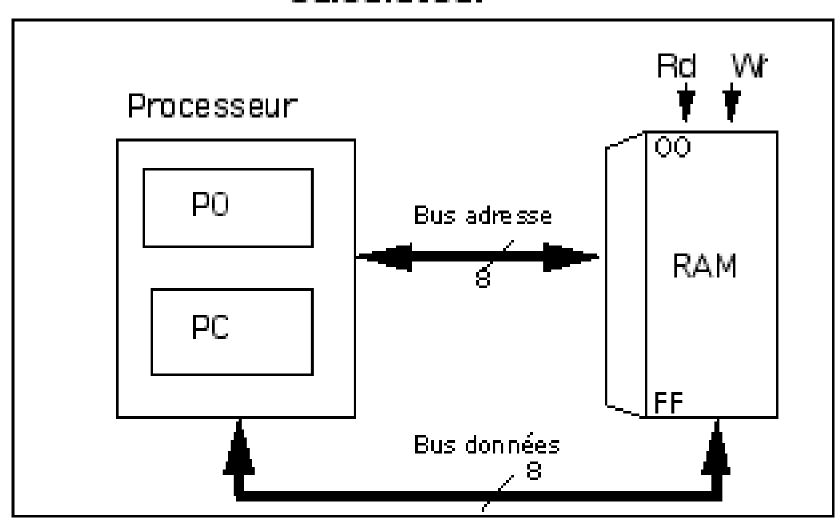
Le micro-langage est donc un moyen d'écrire (de décrire) de façon textuelle l'automate représentant la partie contrôle de la machine.
Une micro-instruction a un sens physique : elle correspond à un (quelquefois 2 ou 3) transfert de registres qui s'exécute en 1 période d'horloge, c'est-à-dire en un cycle de base de l'évolution de l'état interne de la machine.

Il importe dès maintenant, de bien comprendre les différences entre le langage machine du processeur étudié et le micro-langage qui sert à écrire l'algorithme d'interprétation des instructions.

Pour réaliser et tester ces microprogrammes, vous utiliserez l'environnement appelé ProceSim (outil pédagogique de simulation de processeur prévu à cet effet). Les microprogrammes que vous écrirez, pourront être traduits en micro-code (binaire) grâce à un micro-assembleur.
Vous testerez ensuite vos microprogrammes en observant l'état de la partie opérative d'un processeur donné au fur et à mesure du déroulement du microprogramme.

## 2) La machine vue par le programmeur.

On considère un calculateur composé d'un processeur et d'une RAM
Il contient un processeur, 256 mots de 8 bits de mémoire vive (RAM) qui occupent les adresses de $00_H$ à $FF_H$




Le programme qui va s'exécuter sur le microprocesseur et ses données sont stockés dans la RAM.

**Taille des adresses et des données :**
Les adresses et les données sont des mots de 8 bits.

**Registres accessibles au programmeur :**
6 registres banalisés (de 8 bits) : A, B, C, D, E, Ac

3 registres spécifiques :
1- F (mot d'état),
2- SP (pointeur de pile),
3- PC (compteur programme)

A la mise sous tension, PC <- $00_H$, F <- $00_H$, SP <- $00_H$.
Si l'on veut initialiser le pointeur de pile à une autre valeur, il faudra soit le faire par programme, soit par microinstructions (voir plus loin).

**Registre connu, mais non explicitement accessible par les instructions :**
1 Registre instruction : IR

**Registres inconnus du programmeur** : TIR, MK1, MK2

La compréhension du fonctionnement de la partie opérative et de la partie contrôle doit se faire en parallèle.

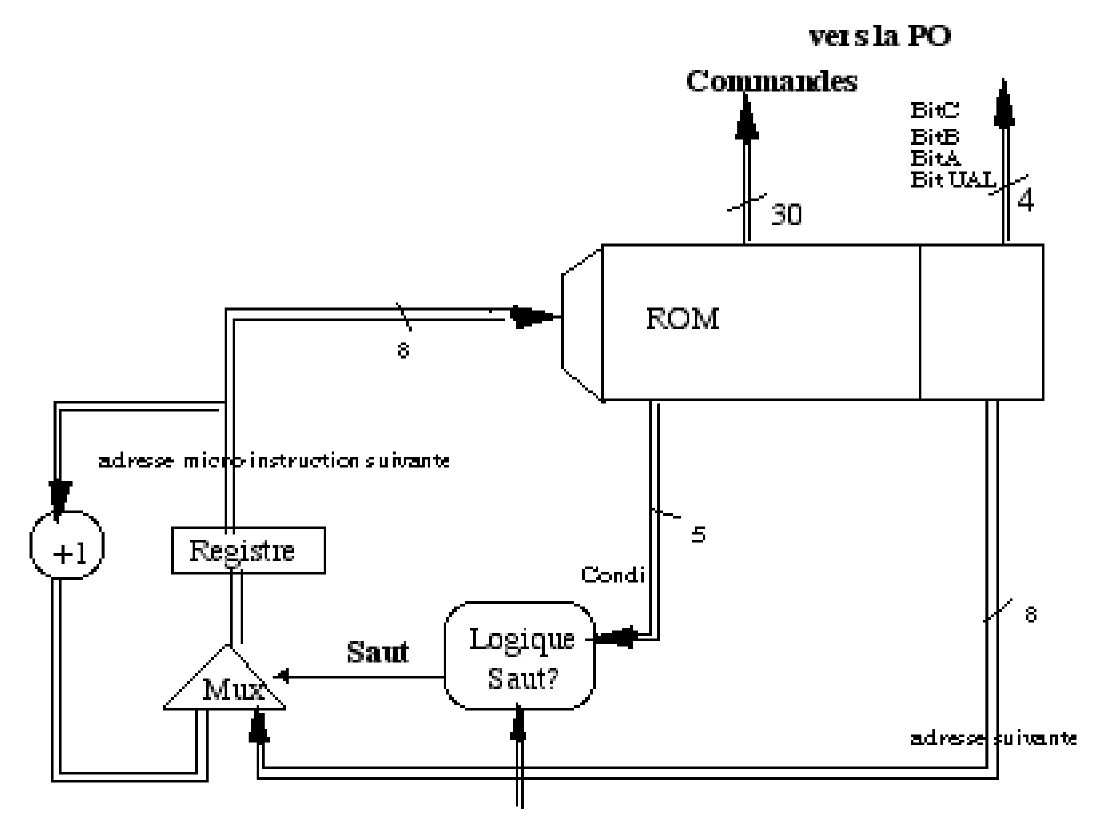
## 3) La partie contrôle du processeur

La partie contrôle est la réalisation d'un automate de Moore grâce à une ROM. Cette technique est appelé microprogrammée. Chaque ligne de la ROM correspond à un état de l'automate et contient les informations nécessaires au calcul de l'état suivant

(nouvelle adresse dans la Rom) et les valeurs des sorties (commande de la partie opérative et commandes extérieures).

---

## Architecture de la partie contrôle:




<u>**Architecture de la partie contrôle micro-programmé**</u>

Les micro-instructions sont stockées dans une ROM.

La boîte appelée "logique saut?" contient le circuit logique combinatoire qui permet, à partir du champ *condi* de la micro-instruction, des micro-Flags et du registre instruction, de déterminer s'il faut, ou non, sauter à la micro-instruction dont l'adresse est dans le champs "adresse suivante".

**Format des micro-instructions:**

Les champs des micro-instructions peuvent être classés suivant deux fonctionnalités.
Certains champs contiennent les valeurs des commandes de la partie opérative dans l'état courant de l'automate de contrôle.
D'autres champs concernent l'enchaînement entre les états de l'automate: si la condition de saut est réalisée (champ *condi*), on passe à la micro-instruction dont l'adresse se trouve dans un champs de la micro-instruction, sinon on passe à la micro-instruction suivante.

| 5 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Condi | Dec | Mux | Mb | Rd | Wr | Ma | Si | Ci | F | UAL | C |

| 4 | 4 | 4 | 8 |
|---|---|---|---|
| BusC | BusB | BusA | Adresse prochaine micro-instruction |

Ou bien

| | 1 | 1 | 1 | 1 |
|---|---|---|---|---|
| | BitUAL | BitC | BitB | BitA |

**COMMANDES:**

Les registres sont numérotés de 0 à 15 dans l'ordre suivant de leurs noms: PC, SP, IR, TIR, 0, 1, ff, A, B, C, D, E, F, MK1, MK2, Ac. (par exemple A correspond au numéro 7). C'est ce numéro qui apparait dans les champs de micro-instruction concernant les registres .
Le registre 0 contient la constante 0, 1 contient 1 et ff contient -1.

**Commandes de sélection :**

**Bus C** (4 bits): si commande non paramétrée, numéro du registre qui sera chargé. (cela peut être le numéro d'un registre constant, mais alors chargement non effectif)

**Bus A**(4 bits) :si commande non paramétrée, numéro du registre qui se trouve sur le bus A.

**Bus B** (4 bits) :si commande non paramétrée, numéro du registre qui se trouve sur le bus B.

**Mux** (1 bit ) : 1 : transmet bus A
0 : transmet MB (sur l'entrée g de l'UAL)

**Commandes de chargement :**

**Ma** (1 bit ): 1 : MA <- ce qui est sur le B-bus
0 : MA inchangé

**Mb** (1 bit ): 1 : MB <- sortie du décaleur
0 : MB inchangé

**Rd** (1 bit) : 1 : lecture. MB <- Mem[MA]

**Wr** (1 bit): 1 : écriture. Mem[MA] <- MB

**C** (1 bit) **:** Autorise/interdit le chargement de ce qui est sur le bus C dans le registre spécifié par le champs Bus C

**Commandes de calcul :**

**Ual** (2 bits): 00: somme arithmétique des 2 entrées de l'UAL
01 : entrée de gauche
11 XOR des 2 entrées de l'UAL
10 : ET des 2 entrées de l'UAL

**Décal** (2 bits) : 00 : rien
01 : sortie ual décalée à droite
10 : sortie ual décalée à gauche
11 : non utilisée

**Si** (1bit) : 1: mettre à 1 le bit I du registre F
**Ci** (1bit) : 1: mettre à 0 le bit I du registre F

**F** (1 bit): 1: mettre à jour les flags du registre F (Z, N et C) par la valeur actuelle des microflags.

**Commandes paramétrées :**

Si la micro-instruction n'utilise pas la partie adresse suivante, il est possible de :

- paramétrer les numéros de registres qui seront : soit sortis sur les bus A et B, soit chargés avec le contenu du bus C. Ainsi, ce ne sont plus les valeurs des champs busA, busB et busC de la micro-instruction qui détermineront les registres à utiliser mais le contenu des registres MK1 et MK2.

**BitC** selection *busc* remplacée par les 4 bits de poids fort de MK1
**BitB** sélection *busb* remplacée par les 4 bits de poids faible de MK1
**BitA** sélection *busa* remplacée par les 4 bits de poids fort de MK2

- paramétrer le calcul à effectuer dans l'UAL

**BitUAL** opération de l'UAL spécifiée dans le champ ***UAL*** remplacé par les bits 1 et 0 de MK2.

---

**CALCUL DE l'ETAT SUIVANT :**

**Condi** (5 bits) :
11 000 :Saut si bit 0 du registre IR =1
11001: Saut si bit 1 du registre IR =1
...
11111: Saut si bit 7 du registre IR =1

10000 : Saut inconditionnel

10110 : Saut si microZ=1
10101 : Saut si microN=1
10100 : Saut si microC=1

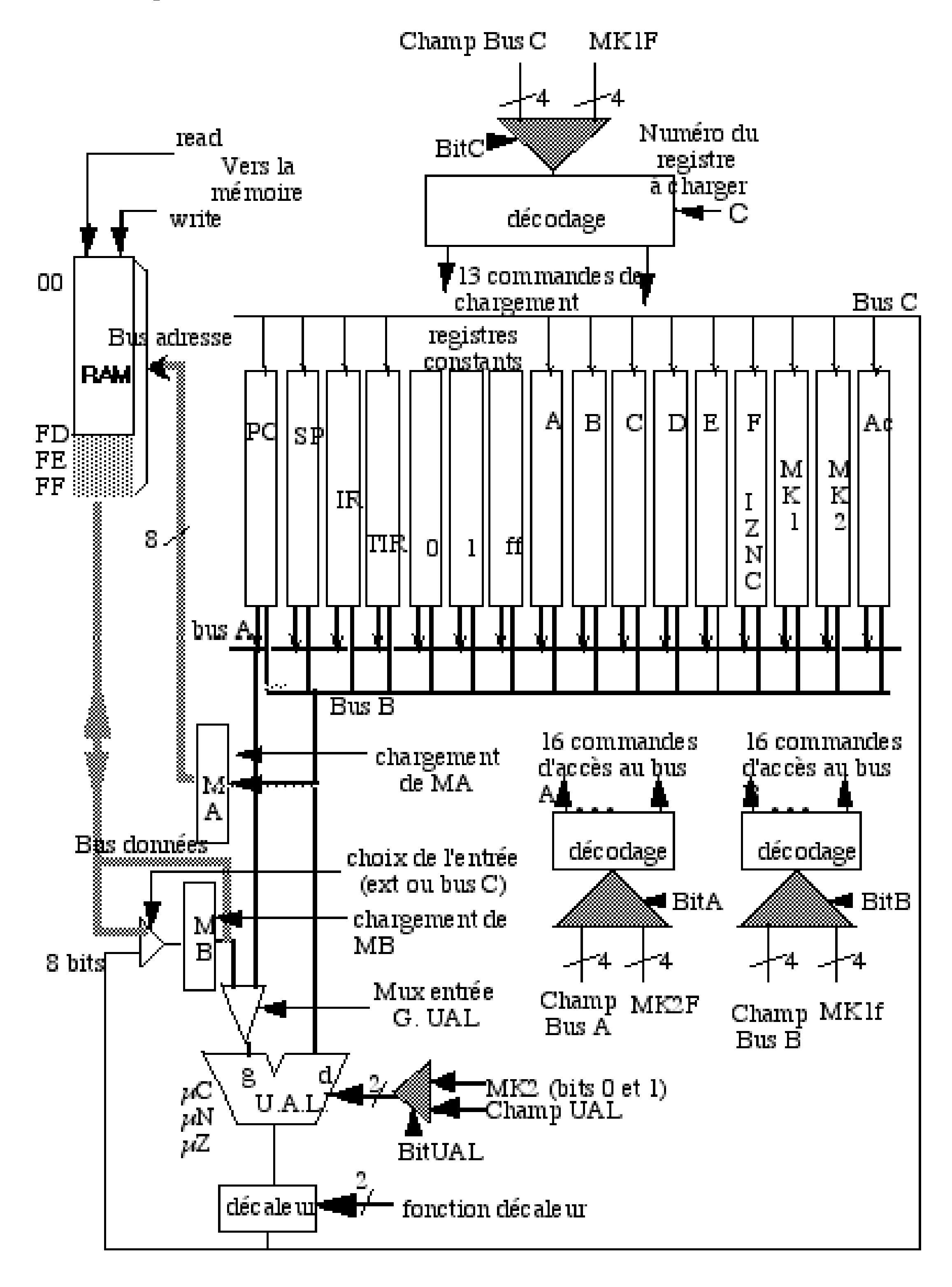
10111: Saut si bit I du registre F = 1
10001: Saut si interruption externe

**Adresse suivante** (8 bits) :
adresse de la prochaine micro-instruction (si la condition de saut est vérifiée).

**Remarque :** Dans le cas d'un saut aucune opération et chargement de registre ne sont réalisés dans la PO (toutes les commandes de chargement de registre sont inhibées). Ce serait tout à fait possible de le faire mais ce n'est pas prévu dans le simulateur et le micro-assembleur (ex: jn etiquette // pc= pc +1).
On ne pourrait pas toutefois utiliser dans ce cas là, les commandes paramétrées puisque les bits BitC, BitA , BitB et BitUAL sont dans le champ *adressesuivante*. Il faudrait de plus « inhiber » ces bits dans la PO par exemple en rajoutant des and avec le dernier bit du champ Condi. (à 0 quand ce n'est pas une microinstruction de saut).

## 4) La partie opérative du processeur

Les commandes correspondent directement aux différents champs des micro-instructions correspondants.




Les multiplexeurs grisés correspondent aux commandes paramétrées.

Les boîtes de décodage permettent d'obtenir 13 ou 16 signaux (chargement de registre ou sélection d'un registre sur un bus à travers une porte trois états) à partir d'un numéro sur 4 bits.

## 5) Primitives pour l'écriture de micro-programmes

Plusieurs actions ont lieu en même temps dans la P.O.
En général

- sortie d'un registre sur le bus A
- sortie d'un registre sur le bus B
- Opération U.A.L.
- Opération décaleur
- Rangement du résultat dans un registre via le bus C.

On écrira :

reg.dest = fct décaleur ( sourceA opérationUAL sourceB)

reg.dest (Destinations possibles) : les 16 registres (y compris 0,1,ff inchangés) , MB, MA, lcbus

| | |
|---|---|
| sourceA : | les 16 registres + MB + labus |
| sourceB : | les 16 registres + lbbus |
| opération UAL : | +, AND, XOR, entrée gauche (MB ou busA) inchangée |
| fct décaleur : | shl, shr, rien (décalages gauche ou droitre de 1 position) |

- Exemples de microinstructions valides (autre que celles du micro-programme)

| dest | decaleur | sourceA | opérationUAL | sourceB | |
|---|---|---|---|---|---|
| **pc =** | | **mb** | **+** | **1** | |
| **tir =** | **shl (** | **ir** | **xor** | **ff )** | |
| **mb =** | | **ac** | | | <opération = entrée gauche> |
| **a =** | **shr (** | **b** | **+** | **lbbus )** | |
| **0 =** | | **ir** | **and** | **1** | |

Les micro-instructions modifient les micro-flags $\mu$Z, $\mu$N, $\mu$C selon les résultats du calcul dans l'UAL. Ils sont mis à jour dans les flags Z N C connus du programmeur (dans le registre F) grâce à la commande **majflag** (champ F de la micro instruction).

**labus** désigne le registre dont le numéro est dans les 4 bits de poids forts de MK2
**lbbus** désigne le registre dont le numéro est dans les 4 bits de poids faibles de MK1
**lcbus** désigne le registre dont le numéro est dans les 4 bits de poids forts de MK1
**lual** désigne l'opération à effectuer dans l'UAL dont le code est dans les bits 1 et 0 de MK2.

**ema** peut être ajouté à toute micro-instruction. Il y a alors chargement de MA par ce qui est sur le bus B. On peut aussi écrire **pc = 1+pc ma=pc**

- micro-branchements :

**j0,j1,...j7**: saut si ième bit de IR est à 1
**jp** saut inconditionnel, **jz** saut si microZ = 1, **jn** saut si microN = 1, **jc** saut si microC = 1
**ji**: saut si bit I du registre F est à 1
**jq** saut si interruption externe

syntaxe: j* étiquette

- Spéciales :

**seti** et **clri :** mise a 1 et à 0 du bit I de F (sert au traitement des interruptions)

- Accés à la mémoire :

**read/write**: permet la lecture /écriture d'un mot en mémoire via les registres MB (donnée) et MA (adresse).
Le temps d'accès de la mémoire est supposé de deux périodes d'horloge du processeur (2 read) pour la lecture et de 3 cycles pour l'écriture (3 write).

## 6) Le jeu d'instructions du processeur

Cette architecture permet de réaliser différents jeux d'instructions en modifiant juste le contenu de la ROM.
Nous allons étudier un jeu d'instruction d'un processeur très simple (jouet) à but pédagogique. Nous n'emploierons qu'une partie des possibilités de la PO et PC vues précédemment.

**Le processeur jouet**

Les instructions sont toutes sur 2 octets:(code opération de l'instruction, opérande).

L'opérande représente soit une adresse (adressage direct), soit une valeur immédiate.

Un seul registre de calcul Acc.

Instructions (code en décimal):

- load immediat (128) : Acc<- Valeur immédiate
- store direct ( 64) : Mem[adresse] <- Acc
- jmp (32) : PC <- Adresse
- add direct (16) : Acc<-Acc+ Mem[adresse]

## Micro programme correspondant à l'exécution du jeu d'instructions jouet

```
! Microprogramme d'un processeur a jeu d'instruction tres simple
! Les instructions sont toutes sur 2 octets:(code operation de l'instructions, operande)
! code des instructions (en decimal): load immediat 64, store direct 96, jmp 128, add direct 32
fetch:        pc=0+pc ema
              read
              read pc=pc+1    ! lecture du codop , PC incremente en prevision du 2eme mot
              ir = mb
              pc=0+pc ema
              read            ! lecture de l'operande,
              read pc=pc+1    ! PC incremente en prevision de l'instruction suivante
              tir = mb        ! tir contient l'operande
decode:       j7 jump         ! si ir7=1 c'est un jmp
              j6 store_load     ! si ir6=1 c'est un store ou un load
              j5 add          ! si ir5=1 c'est un add
              jp fetch        ! sinon c'est une instruction illegale, on ne la traite pas,
                              ! retour au fetch
store_load :  j5 store        ! si ir5=1 c'est un store
```

```
            jp load         ! sinon c’est un load
load:       ac=tir          ! traitement du load immediat:  Acc=2eme mot
            jp fetch
store:      tir=0+tir ema   !traitement du store direct: Mem(2eme mot)= acc
            mb=ac
            write           ! 3 write obligatoire car la memoire est supposee plus lente
            write
            write
            jp fetch
jmp :       pc=tir          ! traitement du jmp: PC =2eme mot
            jp fetch
add :       tir=0+tir ema   ! traitement du add direct : ACC= ACC+Mem(2eme mot)
            read            ! 2 read obligatoire car la memoire est supposee plus lente
            read
            ac=mb+ac
            jp fetch
```

## 7) Mode d'emploi succint de Procesim

* Lancement du simulateur:(Visualisation des transferts de registres = trace de microprogramme ):

**procesim**

• Icône disquette RAM: (fichier suffixé **.ob** )
chargement en Ram du programme objet assemblé s'il existe. Une fenêtre pemet de visualiser le contenu de la RAM.

• Icône disquette ROM (fichier suffixé **.mob**)
chargement en ROM du microprogramme standard fourni ou de celui que l'on a microassemblé. Une fenêtre pemet de visualiser le contenu de la ROM et de suivre l’évolution du microprogramme.

• Pas à pas: le bouton clock permet d’exécuter la prochaine micro-instruction

• Instruction: Le bouton **Instr** permet d’exécuter une suite de micro instructions jusqu’à l’adresse en ROM spécifiée (en héxa). Attention il faut sortir de ce mode par le bouton **cancel** pour pouvoir utiliser d’autres boutons.

• Reset: le bouton **reset** permet une remise à 0 de tous les registres du processeur (comme le ferait une broche reset sur un vrai circuit)

•

* **Micro-assemblage du fichier nomdefichier.muc contenant le microcode**

**masm nomdefichier.muc**

produit le microcode dans le fichier **sortie.mob**

## 8) QUESTIONS :

Pour commencer à comprendre le fonctionnement de ce processeur, vous pouvez utiliser le microprogramme **jouetnew.muc**. Le jeu d'instruction et les codes correspondants sont donnés dans ce fichier.
Ecrire un programme simple dans le langage de ce processeur, l'assembler à la main (traduction en hexadécimal).

- Exécuter le programme en pas à pas grâce au simulateur.
- Donner le graphe de l'automate de la partie contrôle du processeur jouet.
- Comprendre l'effet des commandes de la PO se trouvant dans la ROM
- Donner la fonction booléenne qui correspond à la sortie de la boite "logique saut?" apparaissant dans la figure sur l'architecture de la PC.

On veut étendre le jeu d'instruction de ce processeur jouet.

1- Les instructions ADD et LOAD ont maintenant deux modes d'adressages possibles : direct et immédiat. Rajouter un champ « mode adressage » sur un bit dans le code des instructions. On choisira le bit 3 pour ce champ.
Modifier le microprogramme et tester.
Combien de périodes d'horloge faut-il pour exécuter chacune de ces instructions ?

2- On veut rajouter une instruction équivalente au BL (Branch and Link) du processeur ARM. Cette instruction permet de sauter à l'adresse donnée et de sauvegarder l'adresse de retour dans le registre A. (BL adresse : A=PC ; PC=Adresse).
On rajoute aussi une instruction spécialisée de retour de sous programme RTS qui a pour effet de mettre dans PC le contenu du registre A. Choisir un code pour ces deux instructions. On prendra soins de laisser les bits de poids faible de l'instruction pour traiter la question 4.
Modifier le microprogramme et tester.

3- On veut rajouter une instruction de branchement conditionnel : BLEQ. Cette instruction permet un branchement si le flag Z =1.
Modifier le microprogramme de telle façon que cette nouvelle instruction puisse être exécutée et que l'instruction ADD modifie le flag Z.

4- Rajouter les instructions load et add mais affectant cette fois un des registres A, B, C, D, E, ACC (Ex : ADD E , VI ! E=E+VI, ou en direct ADD B, Ad ! B=B+Mem[Ad]).
Le numéro du registre concerné pourra se trouver dans les 3 bits de poids faible du premier mot de l'instruction. On se servira des micro-instructions « paramétrées » : labus, lbbus et lcbus.

Rédiger un compte rendu comportant :
- Les réponses aux diverses questions posées ci-dessus.
- Le microprogramme modifié (bien commenté)
- Des programmes de test
- Des commentaires sur les choix effectués et des explications sur le microprogramme.